

A Unified Image Processing Framework for Computer Vision and Remote Sensing

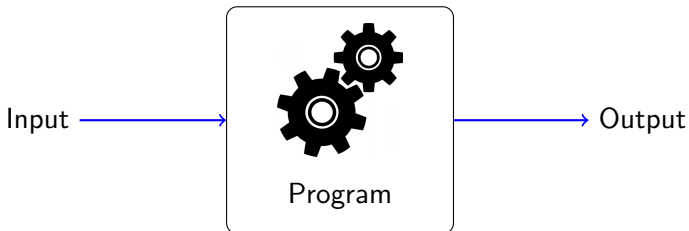
Carsten Brandt, Ludmilla Brandt, Marcus Zepp,
Akarsh Seggemu

July 15, 2015

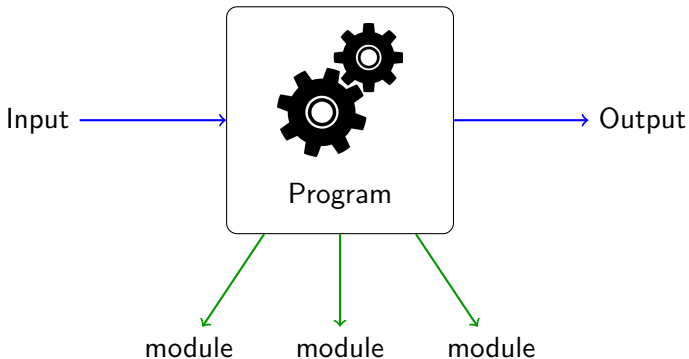
- 1 What is this all about?
- 2 Working with the program
- 3 How to write a new module
- 4 Live Demo

- 1 What is this all about?
- 2 Working with the program
- 3 How to write a new module
- 4 Live Demo

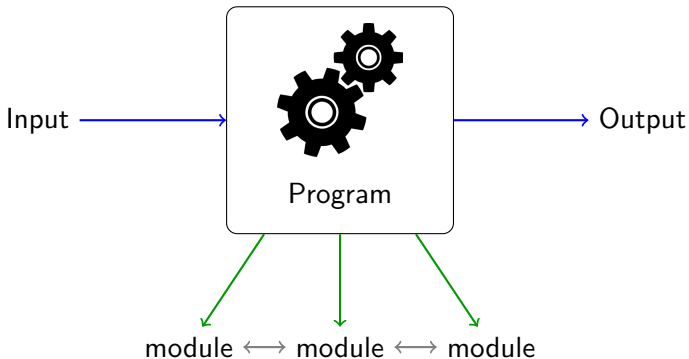
Unified Image Processing Framework



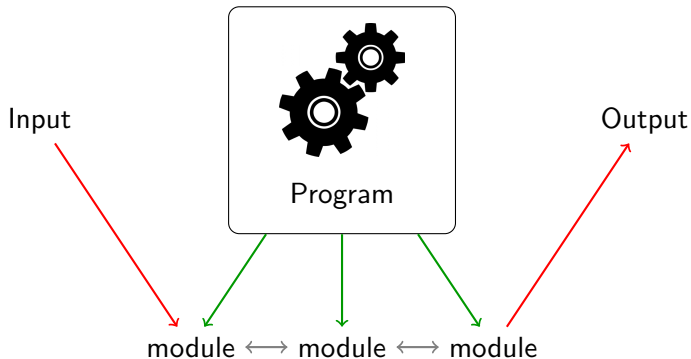
Unified Image Processing Framework



Unified Image Processing Framework



Unified Image Processing Framework



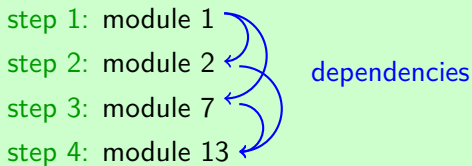
Use given modules

Framework

- module 1
- module 2
- ⋮
- module n

Specify execution order


Chain



Modify the chain

Chain 1

step 1: module 1
step 2: module 2
step 3: **module 7**
step 4: module 13




dependencies

The diagram shows four steps listed vertically. To the right of the list, the word 'dependencies' is written in blue. Four blue curved arrows point from the right side of 'step 3: module 7' to the right side of each of the four steps, indicating that step 3 depends on all previous steps in the chain.

Modify the chain

Chain 2

step 1: module 1
step 2: module 2
step 3: module 24
step 4: module 13




dependencies

Modify the chain

Chain 3

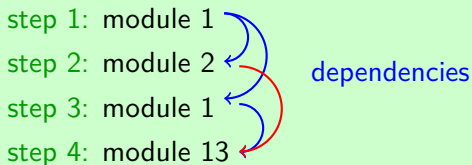
step 1: module 1
step 2: module 2
step 3: **module 1**
step 4: module 13



dependencies

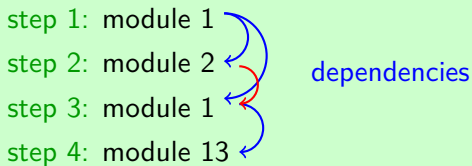
Modify the chain

Chain 4

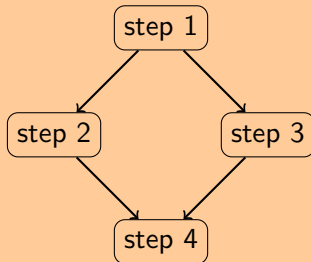


Modify the chain

Chain 5



Visualization



Storage

MyChain xy

step 1: module 1

step 2: module 2

step 3: module 1

step 4: module 13

dependencies

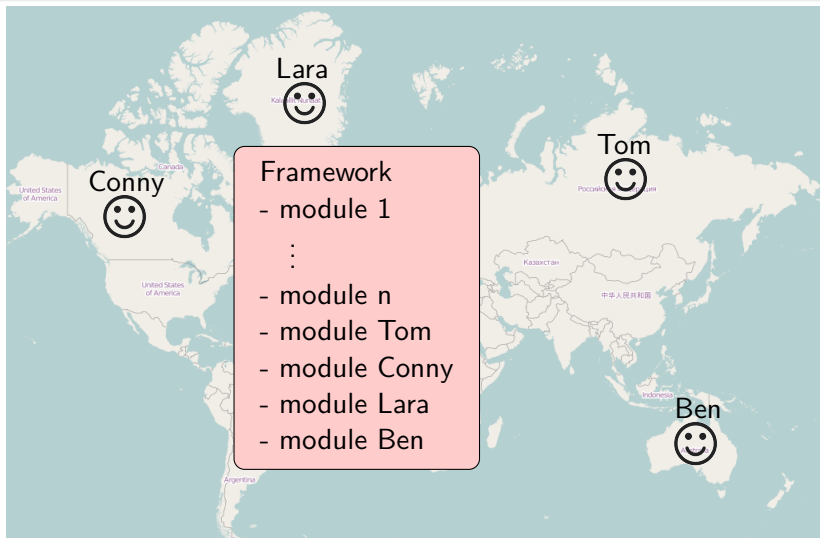


Extend the modules

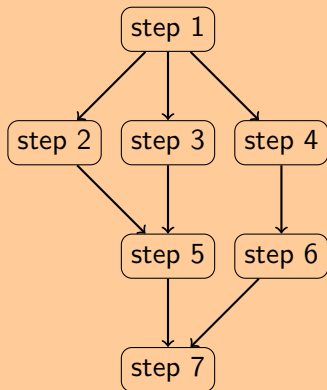
Framework

- module 1
- module 2
- ⋮
- module n
- module n+1
- module n+2
- ⋮

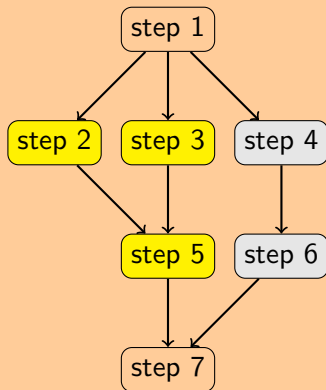
Share the modules



Parallel execution



Parallel execution



Platform independent



Framework

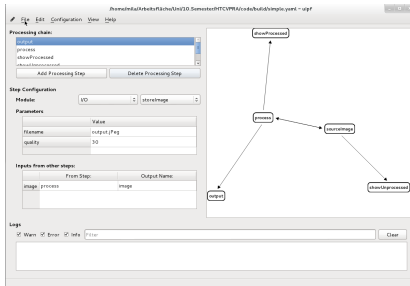


Microsoft Windows

- 1 What is this all about?
- 2 Working with the program
- 3 How to write a new module
- 4 Live Demo

User Interfaces

GUI



Console

```
Usage:
./uipf -c <path to configuration file>                run a processing chain
from a config file.
./uipf -m <moduleName> -i <input> [-p <params>] -o <output>]  run a single module.

Allowed options:

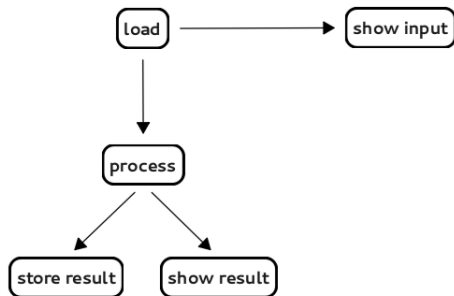
Generic options:
-v [ --version ]      print version string
-h [ --help ]         produce help message

Program options:
-c [ --configuration ] arg defines the path to an already created and stored
                           chain configuration, written in yaml
-i [ --input ] arg      defines an input, can be used multiple times
                           format: inputName:fileName inputName is optional
                           if there is only one input
-o [ --output ] arg     defines an output, can be used multiple times,
                           format: outputName:fileName, outputName is
                           optional, if there is only one input. Output is
                           optional, if there is only one input and one
                           output, the output filename will be chosen from
                           the input name in this case.
-p [ --param ] arg     defines a parameter, format: name:value
```

General Concepts

Processing Chain

→ Processing Step

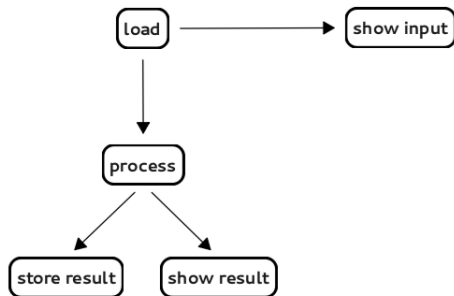


General Concepts

Processing Chain

→ Processing Step

→ Module



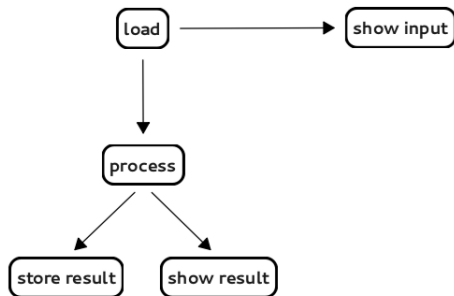
General Concepts

Processing Chain

→ Processing Step

→ Module

→ Parameters

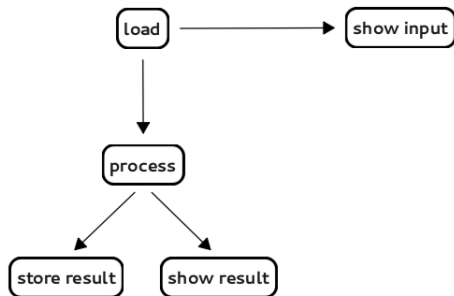


General Concepts

Processing Chain

→ Processing Step

- Module
- Parameters
- Dependencies

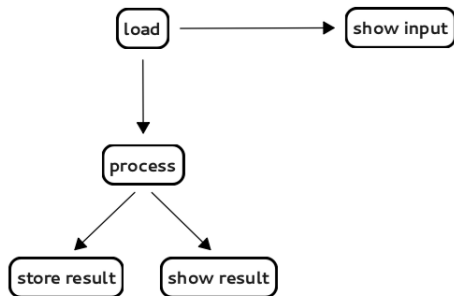


General Concepts

Processing Chain

→ Processing Step

- Module
- Parameters
- Dependencies



Storage format: YAML

```
1 input:
2   module: loadImage
3   filename: input.png
4
5 gauss:
6   module: gaussian
7   input:
8     image: input.image
9   sigmaX: 10
10  sigmaY: 15
11
12 show:
13   module: showImage
14   input:
15     image: gauss.image
```



Storage format: YAML

```
1  input:
2    module: loadImage
3    filename: input.png
4
5  gauss:
6    module: gaussian
7    input:
8      image: input.image
9    sigmaX: 10
10   sigmaY: 15
11
12  show:
13    module: showImage
14    input:
15      image: gauss.image
```



Storage format: YAML

```
1  input:
2    module: loadImage
3    filename: input.png
4
5  gauss:
6    module: gaussian
7    input:
8      image: input.image
9    sigmaX: 10
10   sigmaY: 15
11
12  show:
13    module: showImage
14    input:
15      image: gauss.image
```



Storage format: YAML

```
1  input:
2    module: loadImage
3    filename: input.png
4
5  gauss:
6    module: gaussian
7    input:
8      image: input.image
9    sigmaX: 10
10   sigmaY: 15
11
12  show:
13    module: showImage
14    input:
15      image: gauss.image
```



Storage format: YAML

```
1  input:
2    module: loadImage
3    filename: input.png
4
5  gauss:
6    module: gaussian
7    input:
8      image: input.image
9    sigmaX: 10
10   sigmaY: 15
11
12  show:
13    module: showImage
14    input:
15      image: gauss.image
```



Storage format: YAML

```
1  input:
2    module: loadImage
3    filename: input.png
4
5  gauss:
6    module: gaussian
7    input:
8      image: input.image
9    sigmaX: 10
10   sigmaY: 15
11
12  show:
13    module: showImage
14    input:
15      image: gauss.image
```



The Graphical User Interface

/data/cebe/Dokumente/Uni/htcv-pj-a/examples/convolution.yaml - uipf

File Edit Configuration View Help

Processing chain:

- kernel
- result
- show image
- show kernel

Add Processing Step Delete Processing Step

Step Configuration

Module: I/O storeImage

Parameters

	Value
filename	output.png
quality	

Inputs from other steps:

	From Step:	Output Name:
image	convolution	image

Logs

☒ Warn ☒ Error ☒ Info Filter

Clear

```
graph TD; kernel --> convolution; image --> convolution; convolution --> show_kernel[show kernel]; convolution --> show_image[show image]; convolution --> result; show_image --> show_result[show result];
```

The Graphical User Interface

Processing chain:

kernel
result
show image
show kernel

Add Processing Step Delete Processing Step

Step Configuration

Module: I/O storeImage

Parameters

	Value
filename	output.png
quality	

Inputs from other steps:

	From Step:	Output Name:
image	convolution	image

The Graphical User Interface

Processing chain:

kernel
result
show image
show kernel

Add Processing Step Delete Processing Step

Step Configuration

Module: I/O storeImage

Parameters

Parameter	Value
filename	output.png
quality	

Inputs from other steps:

	From Step:	Output Name:
image	convolution	image

Logs

☒ Warn ☒ Error ☒ Info Filter Clear

Diagram illustrating the processing flow:

```
graph TD; kernel --> convolution; image --> convolution; convolution --> result; convolution --> show_result[show result]; convolution --> show_image[show image]; kernel --> show_kernel[show kernel];
```

The Graphical User Interface

The screenshot displays the ULPF (User Interface for Processing Framework) application. The main window is titled `/home/mila/Arbeitsfläche/Uni/10.Semester/HTCV/PRA/testdata/ConvAndGauss.yaml - ulpf`. It features a menu bar with `File`, `Edit`, `Configuration`, `View`, and `Help`.

Processing chain: A list on the left contains `conv`, `gaus`, `load`, and `show`. Below this list are buttons for `Add Processing Step` and `Delete Processing Step`.

Step Configuration: A section for configuring the selected step. It includes a `Module` dropdown, a `Parameters` table with `Value` columns, and an `Inputs from other steps:` section with `From Step` and `Output Name` fields.

Processing Chain Diagram: A central area showing a directed graph of steps. The steps are represented by colored boxes: `show` (blue), `gaus` (green), `load` (green), and `conv` (white). Arrows indicate the flow: `show` points to `gaus`, `load` points to `gaus`, and both `gaus` and `load` point to `conv`.

Logs: A section at the bottom left with checkboxes for `Warn`, `Error`, and `Info`, a `filter` input field, and a `Clear` button. The log text shows: `Done with step 'gaus'.`, `Running step 'show'...`, and `Press any key to continue...`.

Preview Window: A window titled `show` on the right displays a preview of the processed image, which is a landscape with a red field and a tree under a blue sky.

The Graphical User Interface

/data/cebe/Dokumente/Uni/htcv-pj-a/testdata/ConvAndGaussCircularDependency.yaml - uipf

File Edit Configuration View Help

Processing chain:

- gaus
- load
- show

Add Processing Step Delete Processing Step

Step Configuration

Module: I/O showImage

Parameters

	Value
blocking	
title	

Inputs from other steps:

	From Step:	Output Name:
image	conv	image

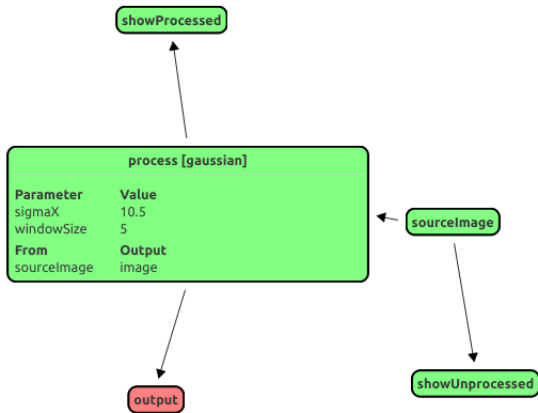
Logs

☒ Warn ☒ Error ☒ Info Filter Clear

There are configuration errors!
Circular dependency detected between the following configuration steps: conv, conv2

```
graph LR; load --> gaus; gaus --> conv; gaus --> conv2; conv --> show; conv <--> conv2;
```

Interface features: Graph



The Console Interface

```
./uipf -c myconfig.yaml
```

```
./uipf <moduleName> -i <input>  
                        -p <params> ...  
                        -o <output>
```

The Console Interface

```
./uipf -c myconfig.yaml
```

```
./uipf <moduleName> -i <input>  
                        -p <params> ...  
                        -o <output>
```

The Console Interface

```
./uipf gaussian -i input.jpg -p sigmaX:5
```

```
./uipf gaussian -i input.jpg -p sigmaX:5  
-o notdefault.png
```

The Console Interface

```
./uipf gaussian -i input.jpg -p sigmaX:5
```

```
./uipf gaussian -i input.jpg -p sigmaX:5  
-o notdefault.png
```

```
./uipf convolution -i image:input.jpg  
-i kernel:kernel.png  
-p sigmaX:5 -o out.png
```

The Console Interface

```
./uipf gaussian -i input.jpg -p sigmaX:5
```

```
./uipf gaussian -i input.jpg -p sigmaX:5  
-o notdefault.png
```

```
./uipf convolution -i image:input.jpg  
-i kernel:kernel.png  
-p sigmaX:5 -o out.png
```

- 1 What is this all about?
- 2 Working with the program
- 3 How to write a new module**
- 4 Live Demo

Basics

Modules are precompiled extensions that:

- encapsulate functionality which can be used in processing steps
- are binary files, shareable without sourcecode (QTPlugin)
- implement a simple interface
- can include own libs as they need them
- have Metadata displayed in the GUI

Interface

Basic interface:

```
1 string name();  
2 void run(DataManager& data);  
3 MetaData getMetaData();
```

MetaData:

```
1 string, // general verbal description of the module  
2 string, // category  
3 DataDescriptionMap, // input  
4 DataDescriptionMap, // output  
5 ParamDescriptionMap // params
```


Interface

Basic interface:

```
1 string name();  
2 void run(DataManager& data);  
3 MetaData getMetaData();
```

MetaData:

```
1 string, // general verbal description of the module  
2 string, // category  
3 DataDescriptionMap, // input  
4 DataDescriptionMap, // output  
5 ParamDescriptionMap // params
```

run() method

```
1 void LoadImageModule::run( DataManager& data) const
2 {
3 // (1) get inputs and params:
4 //     - data.getInputData(inputName)
5 //     - data.getParam (paramName, dafaultValue)
6 // (2) work with them
7 // (3) create output:
8 //     - data.setOutputData(outputName,
9 //       outputContent);
9 }
```

getMetaData() method

```
1  Metadata ResizeModule::getMetaData() const
2  {
3      DataDescriptionMap input = {{"image",
4      DataDescription(MATRIX, "the image to resize.") }
5      };
6
7      DataDescriptionMap output = {{"image",
8      DataDescription(MATRIX, "the result image.") }};
9
10     ParamDescriptionMap params = {
11         {"width", ParamDescription("new width") },
12         {"height", ParamDescription("new height") }
13     };
14
15     return Metadata("Resizes an image using
16     openCV.", "Image Processing", input, output, params );
17 }
```

CMake

Modules need to be registered in CMakeList.txt:

```
10  ...
11  #Image Processing
12  #ResizeModule
13  add_library(ResizeModule SHARED
               modules/improc/ResizeModule.cpp)
14  qt5_use_modules(ResizeModule Core ) #QtCore is needed
    for <QPlugin>
15  target_link_libraries(ResizeModule opencv_core
                          opencv_imgproc ModuleBase)
16  ...
```

Implementation summary

Steps to create your own Module:

- 1 Copy .hpp and .cpp of an existing module e.g. the DummyModule
- 2 Replace "DummyModule" with your new name

Implementation summary

Steps to create your own Module:

- 1 Copy .hpp and .cpp of an existing module e.g. the DummyModule
- 2 Replace "DummyModule" with your new name
- 3 implement your logic in run()

Implementation summary

Steps to create your own Module:

- ➊ Copy .hpp and .cpp of an existing module e.g. the DummyModule
- ➋ Replace "DummyModule" with your new name
- ➌ implement your logic in run()
- ➍ define your module's metadata

Implementation summary

Steps to create your own Module:

- 1 Copy .hpp and .cpp of an existing module e.g. the DummyModule
- 2 Replace "DummyModule" with your new name
- 3 implement your logic in run()
- 4 define your module's metadata
- 5 edit CMakeList.txt

Implementation summary

Steps to create your own Module:

- ➊ Copy .hpp and .cpp of an existing module e.g. the DummyModule
- ➋ Replace "DummyModule" with your new name
- ➌ implement your logic in run()
- ➍ define your module's metadata
- ➎ edit CMakeList.txt
- ➏ run make

Implementation summary

Steps to create your own Module:

- ➊ Copy .hpp and .cpp of an existing module e.g. the DummyModule
- ➋ Replace "DummyModule" with your new name
- ➌ implement your logic in run()
- ➍ define your module's metadata
- ➎ edit CMakeList.txt
- ➏ run make

- 1 What is this all about?
- 2 Working with the program
- 3 How to write a new module
- 4 Live Demo**

Live Demo

Live Demo!

Fork us on Github:

<https://github.com/TU-Berlin-CVRS/uipf>